A Reactive Object Model for Concurrent Engineering Design

Toan Nguyen

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Reactive Object Model for Concurrent Engineering Design

Toan Nguyen

N° 2692
Octobre 1995

PROGRAMME 3

A reactive object model for concurrent engineering design

Toan Nguyen

N° 2692
Octobre 1995

PROGRAMME 3

Intelligence Artificielle, Systèmes Cognitifs
et Interaction Homme-Machine

A reactive object model for concurrent engineering design

Abstract: A reactive object model is defined to control the design process of evolving objects in concurrent engineering environments. The goal is to support engineers in controlling the result of their design decisions, using a software automaton as a plug-in addition to computer-aided design platforms that support concurrent engineering. The automaton implements a reactive design specification for composite objects with multiple representations, that includes a model for their design process. The synchronous language ARGOS is used to implement the automaton. ARGOS is an imperative language for the specification and verification of reactive systems. It is based on the parallel and hierarchical composition of automata.

Key-words: Intelligent Information Systems, Engineering Design, Knowledge Representation and Integration, specification, verification, reactive systems, concurrent engineering, CAD/CAM.

Un modèle réactif pour la conception en ingénierie concourante

Résumé : Un modèle réactif est proposé pour la conception d'objets évolutif en ingénierie concourante. L'objectif est d'aider les concepteurs à contrôler le résultat de leurs décisions de conception à l'aide d'un automate. Ce dernier est conçu comme une extension à une plate-forme logicielle d'ingénierie concourante. L'automate réalise une spécification de la conception d'objets composites qui inclut un modèle du processus de conception. Le langage synchrone ARGOS est utilisé pour l'implantation de l'automate. ARGOS est un langage impératif de spécification et de vérification de systèmes réactifs. Il est basé sur la composition parallèle et hiérarchique d'automates.

Mots-clés : Systèmes d'information intelligents, Conception, Ingénierie concourante, Ingénierie simultanée, Représentation de connaissances, spécification, vérification, systèmes réactifs, CAO.

1. Motivations

The design of integrated concurrent engineering platforms has received much attention in recent years, because competition strives for shorter design delays and manufacturing costs among competing firms [DER89]. Concurrent engineering allows for parallel design of product components, thus leads to shorter design to market delays. It requires, however, advanced coordination and integration capabilities [BRO92]. One way to assist the design engineers in such environments is to provide flexible collaborative frameworks. They allow various teams to work simultaneously and consistently on different parts of a global project [CUT93, TEN92]. But the existing software, tools and computer platforms used in manufacturing enterprises require powerful, versatile and open architectures to take into account these legacy systems [MCG94].

A potential technological breakthrough consists in developing generic integration platforms that provide high-level distributed services [GEN92b], i.e., at the applications' knowledge-level [NEW82]. This allows the various tools to communicate and cooperate through high-bandwith networks of distributed computers [GEN92a]. These machines offer specific sharable services. Much of this approach is undertaken by the ARPA Knowledge Sharing Effort [PAT92].

A requisite for developing such platforms is the specification of a model for a generic design process, allowing the consistent cooperation of the distributed services.

Yet, the design of such a generic model is impaired by three factors:

- the expertise on which the model can build, does not emerge from the computer science community, but rather from the various application areas,
- history tells that expertise is very difficult to acquire, and to translate into generic concepts that can be used by software engineers for implementation. In fact, this has called during the last decades upon expert systems, AI and applied maths techniques for the least,
- last, the variability and heterogeneity of the various disciplines seem to render unreachable in the short term the definition and implementation of a generic design model. It is likely, indeed, to be immediately contradicted by the specifics of each particular discipline, and by the individual requirements of the field engineers to whom it is intended.

This lead us to explore another approach, based on the monitoring of the evolution of the design objects, following the engineers' design decisions.

The idea is to track closely the design path followed by the engineers, and to provide a suitable reactive model of the design artifacts. The reactive object model is designed to evaluate the consequences of the design actions. The latter are modeled as modification requests, that are sent simultaneously by the various teams working on a concurrent engineering platform.

This approach capitalizes on specification and verification techniques developed for reactive systems [JOU94a]. Specifically, we have designed and developed a monitoring automaton, implemented in the synchronous language ARGOS. ARGOS is an imperative language based on hierarchical and parallel composition of automata [JOU94b, MAR93].

The automaton is designed as a plug-in addition to an integration platform called SHOOD, which is currently being designed at INRIA (Figure 1). SHOOD

aims at providing tools and methods for the integration of engineering design systems [NGU93].

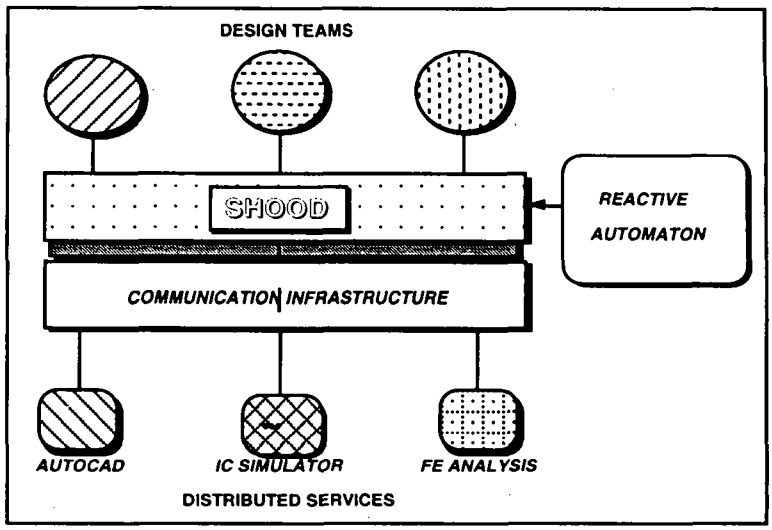

Figure 1. SHOOD: an integration platform for concurrent engineering.

The following section of this paper is an overview of the project SHOOD (Section 2). Section 3 describes the product model that is used to track the evolution of the design objects. Section 4 gives a description of the automaton designed to monitor the design decisions, together with a simple example. Section 5 describes the design process model, based on the combination of asynchronous tasks. Section 6 is an implementation overview of the automaton using the language ARGOS. Section 7 is a conclusion.

2. Overview of SHOOD

The project SHOOD is developed at INRIA, Rhône-Alpes Research Unit, in Grenoble (France). The goal is to provide tools and methods for the integration of engineering design systems. This is developed in three work-packages:

- the definition of a generic product model, based on the object paradigm [NGU92a],
- the formal specification of a reactive design model, which is the main scope of this paper,
- the development of an integration platform, that supports concurrent engineering of the design products (Figure 1).

The generic product model is based on knowledge representation techniques, using a reflexive object model, implementing such concepts as [BOU95b, NGU91]:

- classes, meta-classes, instances, multiple inheritance, specialization,
- composition and dependency relationships (e.g., existential and sharing dependencies, object versions' dependencies),
- generic functions using method specialization,
- active rules, making SHOOD a powerful active-objects store,
- a classification mechanism for composite objects.

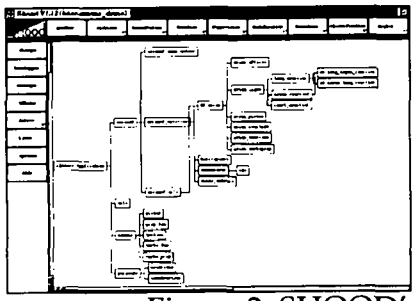

Figure 2. SHOOD's user interface.

Although based on sophisticated concepts, e.g., the reflexive implementation of the object model, an X11-based user interface provides a shallow learning curve to SHOOD. This is implemented by a friendly and easy to use access to object definitions and manipulations (Figure 2).
The product model in SHOOD has been tested so far in engineering design applications, for mechanical simulations, in electric engineering design applications, for the pre-dimensioning of asynchronous electric motors. Using a specific interface, it is used for the management of tender applications, for a federation of 27 contractor companies (Figure 3).

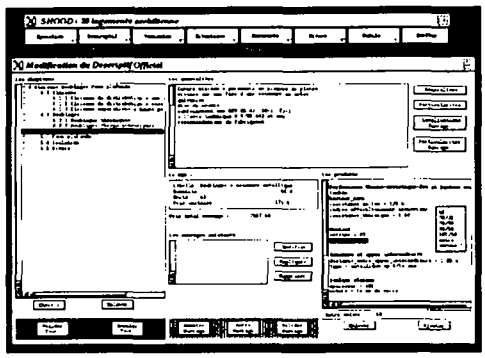

Figure 3. SHOOD tender management.

SHOOD is currently being used for prototyping an intelligent help to a commercial simulator for VLSI circuits, to which it is being interfaced. SHOOD is also interfaced with the AutoCAD™ 3D modeler (Figure 4).
SHOOD is implemented in Le_Lisp™, and includes 50,000 lignes of code. It currently runs on SUN™, Silicon Graphics™, and RISC 6000™ workstations, as well as PC 486 compatible computers.
Current developments include:

- the specification of a communication language between application systems and SHOOD, through a knowledge-level communication protocol, *a la* KQML [FIN92],
- the specification and implementation of an ontology toolbox, called SHOT [BOU95].

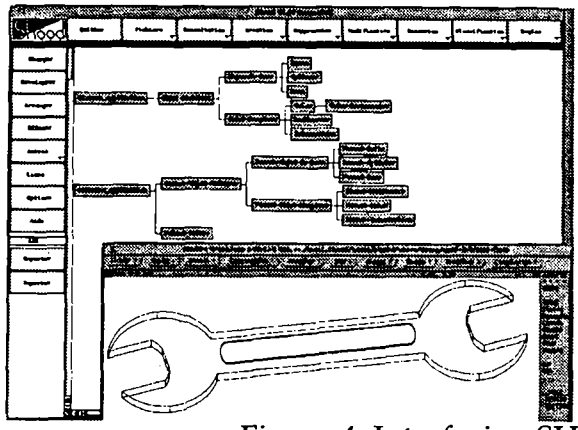

Figure 4. Interfacing SHOOD and AutoCAD™.

3. The reactive object model

The reactive object model implemented by SHOOD supports three basic requirements of engineering design applications [NGU91]:

- the evolution of design components during the design of the products,
- the multiple representations of the objects being designed concurrently by various teams [NGU92b],
- the complexity of the product structures, often considered as intricate "part-of" relationships, but considered also here as tightly inter-dependent components. It entails that semantic relationships, e.g., existential dependency relationships among components, are considered.

These three perspectives are implemented in the object model as particular object classes (Figure 5). Altogether, they define a static product structure, which may be incomplete and inconsistent at any point during the design process. The evolution of the objects is formally defined by a set of rules that specify the permitted transitions of their components, along each particular perspective, and among all three perspectives together [BOU95a, BOU95b].

Considering one perspective, rules may state for example that no component may refer to itself, or may exist independently of any other. This deals with intra-perspective consistency.

Other rules may constrain the sharing of design components to specific states. Usually, the evolution of individual components is restricted to three different states, e.g., public, private and transient [KIM90]. Rules constraining the evolution of the components among the three states can be stated as follows:

- public components cannot be made transient or private, i.e., they have to be duplicated before,
- private components cannot be made transient or public if they include other private components,
- transient components cannot be made public if they include other transient components,
- public components cannot include transient or private components,
- transient components cannot include private components,
- private components may include public and transient components.

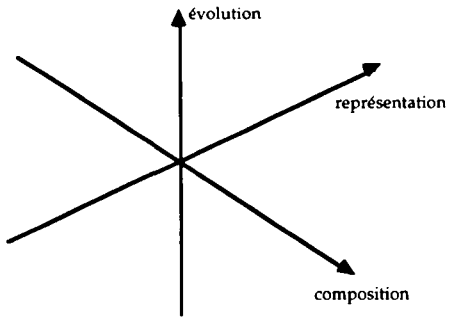

Figure 5. The three modeling perspectives in SHOOD.

Interactions among perspectives are also considered. This deals with notification and compatibility issues. This is out of the scope of this paper.
The objects evolve following the design decisions of the engineers during the design process. The purpose of the automaton is thus to control that the objects' evolution is compliant with the set of rules defining the desired artifacts, and with the specifics of each particular expertise involved in the design.
These rules are usually encoded in application programs, in CAD tools and expert systems. A model of such rules is therefore theoretically feasible. Yet, practical considerations render this approach unrealistic. There are considerable quantities of such programs, tools and systems. Their specifications are usually protected by copyright laws, and their number and heterogeneity make the task of modeling them titanic.

The approach taken here departs from this burdening task. It considers only the evolution of the design objects at any point in time, to track potential discrepencies with design constraints. It is designed to provide an on-line help to the designers. This help is provided by an automaton, which tracks the design path followed by the engineers. It detects any forbidden design step in real-time.
This is achieved by the design of a reactive object model, plugged into the concurrent engineering platform. It provides immediate warnings on hazardous design decisions.
Basically, the automaton provides a reactive model of the design objects, not a model, nor a formalization, of the design methodology. This would be quite challenging to implement, because the various design decisions are globally indeterministic, and the various expertise difficult to capture.
In contrast, the automaton monitors the correctness of the design decisions. It reacts to their resulting effects on the objects being modified. It is intended to run in background on concurrent engineering platforms, such as SHOOD, to provide a reactive help, consisting of real-time warnings to the designers.
The automaton is thus a designer's advisor. It is a verification tool that controls the evolution of the design objects. It is not a model of the design process, although it implements much of the requirements for the specification of a process model. Yet, it is not possible to prove it to be complete, with respect to a specific design methodology.

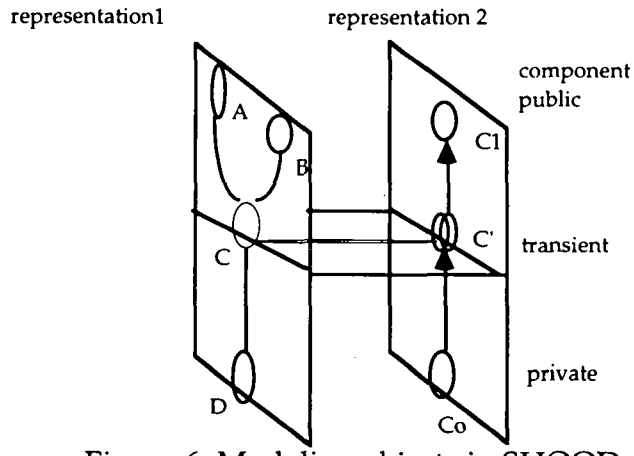

Figure 6. Modeling objects in SHOOD.

So far, it can monitor the evolution of complex composite objects that include simultaneous representations (Figure 6). Designers can work in parallel on these representations, thus making the automaton a good support for concurrent engineering. The automaton monitors in real-time the impact of the designers' modifications on the objects. The concurrent engineering platform is therefore a true reactive system, which tracks closely the design process of complex objects.

The objects are modeled in SHOOD by their different representations [NGU92a]. They are also characterized by their current state (public, private or transient), and their various components.
The representations include specific constraints, called intra-representation constraints, that model design knowledge, particular to the various skills required in the projects. They are also linked by various inter-representation constraints, that model the consistency rules that must hold among the various parts of the projects.
Let O be the set of all the design objects, and let "o" be a particular instance of a design object: o I O.
Let C be the set of all the objects' components. The set is partitioned strictly in three subsets P, A, E of public, private and transient objects, respectively:

$$C = P \vee A \vee E, \text{ and } P \vee A \vee E = O.$$

Let S be the set defined by: S = {public, private, transient}; s_o denotes the current state for object o, where s_o I S. A function state(o), defined on O v C with values in S, delivers the current state of object or component "o":

$$A\ o\ I\ O \vee C,\ E\ s_o\ I\ S : state(o) = s_o.$$

Let R be the set of all object representations, e.g., functional, structural, ... An object instance "o" exhibits a set Ro of representations: $R_o = \{R_1, R_2, \ldots, R_p\}$, such that R_o I 2^R, where 2^R is the powerset of R.
Let I be the set of all intra-representation constraints. $I(R_{i,o})$ denotes the set of all the constraints relevant to the representation R_i for object "o", o I O. I_o is the set of all constraints involving the representation set Ro.
Let T be the set of all inter-representation constraints. Let $T(R_i, R_j)$ be the set of constraints involving the representations R_i and R_j. We consider here only constraints involving two representations simultaneously.

We denote $T(R_i, R_j, o)$ the set of constraints relating the representations R_i and R_j for the object o I O. We have $T(R_i, R_j, o) \sqrt{} T$. We denote also T_O the set of all inter-representation constraints for object "o".

Let C_O be the set of object o's components: $C_O \sqrt{} 2^C$. A function comp(o), defined on O with values in 2^C, delivers the set C_O of the object's components:

$$A\ o\ I\ O, E\ C_O\ I\ 2^C: comp(o) = C_O.$$

The set O is thus defined on C x R x S x I x T, where "x" denotes the cartesian product of two sets: $O \sqrt{} C \times R \times S \times I \times T$.

A particular object instance o I O is thus defined by:

$$o = (C_O, R_O, s_O, I_O, T_O).$$

A component "c" for object "o", c I C_O, o I O, bears a set Rc of representations. A function rep(c), defined on C with values in R, delivers the set Rc to which the component "c" belongs:

$$A\ c\ I\ C, E\ Rc \sqrt{} R : rep(c) = Rc.$$

The set Ro of representations for an object oI O, that includes a set Co of components, is therefore: $Ro = \underset{c}{\vee} Rc$, for all cI C_O.

o	R1	R2	...	Rp
c1	x	x		
c2		x		x
...	...	...	...	...
cn	x			x

Table 1. Object and components' representations.

In Table 1, the representations are outlined by an "x" mark in the cell designated by the line "c_i" and the column "R_j", indicating that component "c_i" includes the representation "R_j". A cell without an "x" mark indicates that the corresponding representation is irrelevant for the component.

The object "o" includes n components c1, c2, ..., cn:

$$Co = \{c1, c2, ..., cn\}.$$

The components include in turn p representations R1, R2, ..., Rp. For example, component c1 includes two representations: R1 and R2. Component c2 bears the representations R2 and Rp, and so forth. The set Ro of all the representations for object "o" is therefore the union of all the representations R1 to Rp for its components: $Ro = \{R1, R2, ..., Rp\}$.

4. The automaton

In order to be useful, the automation must provide a specific help for any on-line decisions that impacts on the object structure. This implies that all the component structures must be considered by the automaton. Thus, all three perspectives defined in the previous section (Section 3) are modeled by the automaton:

- the component structure of the design objects is considered by modeling their evolution among public, private and transient states, within each particular representation of the product. A specific automaton is dedicated to the monitoring of the components' evolution, through the various public, private and transient states defined in the previous section (Figure 7).

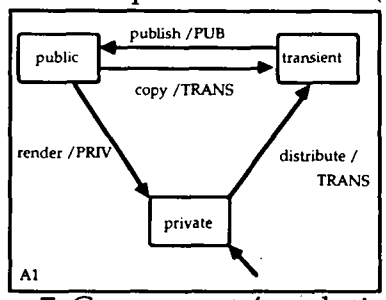

Figure 7. Components' evolution.

- the representations are modeled in the automaton by various automata, one automaton being dedicated to monitor the evolution of each particular representation. Therefore, there are as many automata than there are representations required by the design of a particular product. These representations are not likely to change during the design project. They are therefore characterized at the beginning of the design project.
- the internal constraints within each representation are controlled by specific automata, every constraint being monitored by a simple automaton.
- the constraints relating various representations represent inter-representation constraint propagations. They consider the impact of changes among the diverse object representations. They are monitored by a specific automaton, which is itself composed of individual constraint-monitoring automata.

Let us consider an example composite object O that includes two parallel representations R1 and R2. The representations include the following components for O (Figure 8):

- O1, C11, C12 and C13 for R1,
- O2, C21 and C22 for R2.

The states of the components are the following:

- public components: C11, C12 and C21,
- private components: C13 and C22,
- transient components: O1 and O2.

The following constraints are defined:

- T0 between C11 and C12: C11 < 2 * C12,
- T1 between C21, C12 and C12: C21 = C12 * C11,
- T2 between C13 and C22: C13 = C22.

Let us call CONSTRAINT0, CONSTRAINT1 and CONSTRAINT2 the automata in charge of the monitoring of the constraints T0, T1 and T2 respectively.
Let us call R1 and R2 the automata in charge of monitoring the evolution of each object representation, respectively. They are defined as parallel automata because the two representations are simultaneous point of view on the object O.
Let us call G the global automaton in charge of the control of the object evolution.
G takes as input the requests "màj" for modification of object O. G outputs a boolean value ("accepted" or "rejected") depending on the acceptance or

rejection of the request. This is based on the evaluation of the various constraints T0, ..., T2 by the other automata.
G has two states, i.e., "idle" and "modif", depending on the current evaluation of existing constraints. It loops on the state "idle", waiting for an update request to object O.

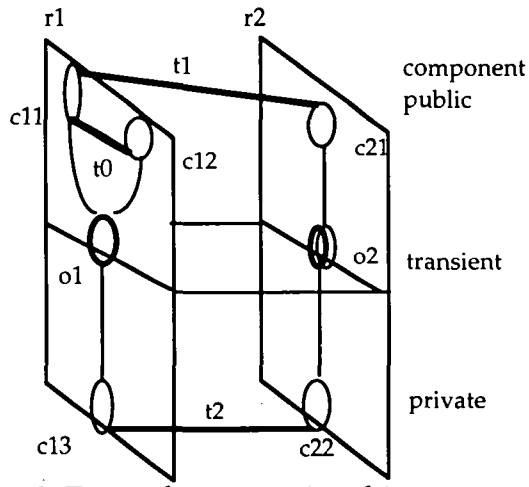

Figure 8. Example composite object.

The transition to state "modif" is fired when a request is issued by the user. It generates the input "màj". Depending on the result of the evaluation of the constraints T0, T1, and T2 by the inner automata, G enters the "idle" state back. It uses for this the inputs "yes" or "no", depending on the evaluation of the constraints. It then outputs the events "accepted" or "rejected" respectively (Figure 9).
The state "modif" is controlled by an automaton that is monitoring the components' state evolution, i.e., from "private" to "transient" and "public". It also controls the modifications performed by the designers on the various object representations.

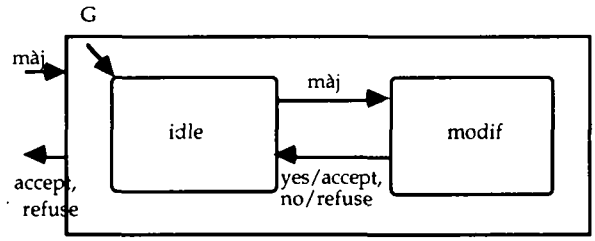

Figure 9. The global automaton.

These two functions are implemented by two parallel automata, i.e., one automaton, called "ETAT", controlling the state evolution of components (Figure 5), the other, called "OBJET", controlling the modifications performed on the representations.
These modifications on the representations are in turn controlled by the parallel decomposition of an automaton, split in two automata, one for controlling each representation. They are called "ModR1" and "ModR2" in Figure 10.

Each automaton, ModR1 and ModR2, includes three states. Their initial state (called "StableI") is an inconsistent stable state. This means that, by default, components are in an inconsistent and incomplete state.

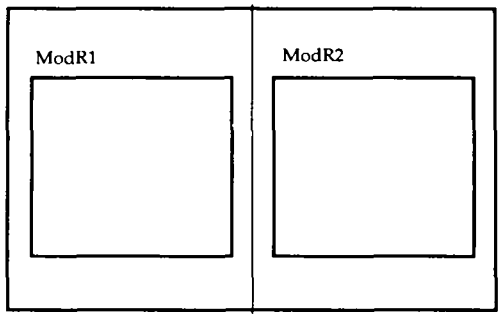

Figure 10. Parallel composition of automata.

Transitions through the various automata controlling the evaluation of the design constraints, called "CONTRAINTE0", ..., "CONTRAINTE2", lead eventually to a stable and consistent state, called "StableC" (Figure 11). It is assumed also that there is no direct transition between the two states "StableI" and "StableC". A modification of the component is requested prior to this transition. There is therefore a "busy" state, called "Encours". This is a means to model the transient state of the objects, for which modifications are pending. This does not make any assumptions on their particular degrees of consistency and completeness [NGU91].
This approach is implemented because the design applications require the characterization of the objects' evolution during the design process. First, their components are incomplete most of the time, until a full specification is reached by the designers. Second, they are also inconsistent most of the time during the design process, because the design constraints are often temporarily violated, until a satisfactory solution has been achieved.

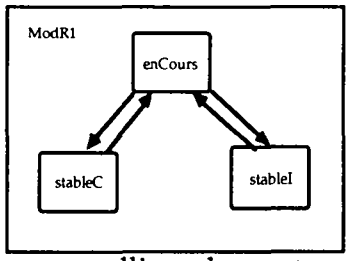

Figure 11. Automaton controlling elementary design constraints.

Finally, the requests for component modifications are processed in sequence by the global automaton G. It entails that even in concurrent engineering environments, where designers work in parallel on the object representations, the automaton described above (Section 4) schedules the requests on a first-in/first-out basis. Updates may be processed in parallel on the individual components, but merging their results must be controlled, or even negotiated. This problem is out of the scope of this paper. We assume here that the global automaton G relies on a coordination model provided by the concurrent engineering platform [PET92]. Therefore, we implemented a centralized and

sequential scheduling policy for the control of the update requests by G. This does not impact on the parallel submission of update requests by the designers, and their verification by the parallel automata ModR1 and ModR2. Still, the interactions between the modifications requested on the parallel representations are handled sequentially.
This approach does not impair the execution of the concurrent engineering environment, because the automaton reacts to the update requests in real-time. It implements a true reactive model of the design objects. This follows from two considerations. First, the requests are processed on-line by the automaton, which runs as a background process, plugged into the engineering platform. Second, using ARGOS, the synchronous language implementation of the automaton guarantees real-time answers to the designers.

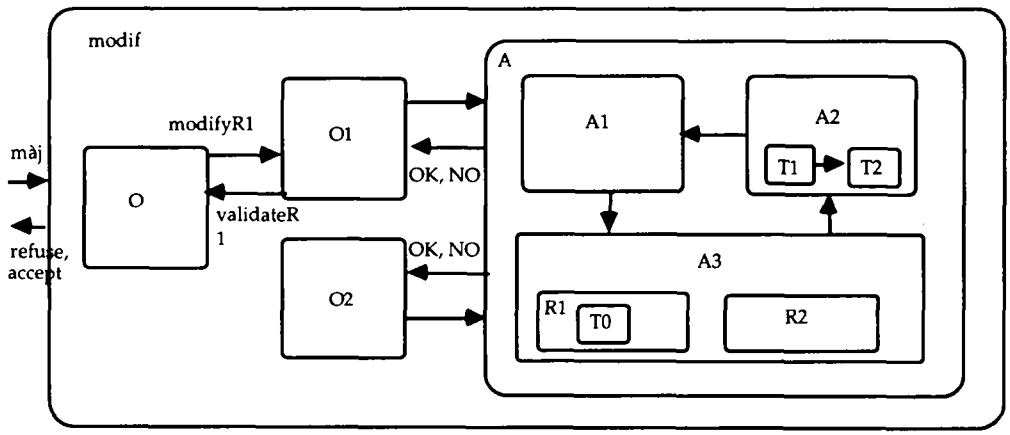

Figure 12. The global automaton.

5. Modeling the design process

Modeling the design process requires extensive knowledge on the activities involved, and a formal background on which to elaborate a suitable model [GER93]. Yet, design decisions can be erratic and amenable to multiple trial and error cycles. Also, the intrisic dynamics of the design process makes it difficult to comprehend and control them using formal techniques. VLSI circuit design is a leading activity in this respect, because high level definition languages are available.

The process model developed for SHOOD is intended to provide a powerful and flexible framework for capitalizing on design experience and achieve new engineering projects. It supports incremental construction of process models and their dynamic modification to adapt to the projects' specific goals. It is based on partially ordered sets of tasks and scheduling operations.

The tasks are defined recursively, i.e., one task may be composed of a partially ordered set of sub-tasks. Terminal tasks call for basic design operations, e.g., 3D modeling primitives. They call for specific software to be activated, using an appropriate interface. Tasks have input and output parameters that collect the information to be processed and the results produced after execution.

The global design process is itself a partially ordered set of tasks. It defines a template for process specification and verification, which can be amended dynamically by the designers. The template is built incrementally using experience gained from previous design projects. It helps therefore to capitalize knowledge and experience. It is also a powerful tool to store and maintain enterprise expertise.

The operations performed on the tasks are control and synchronization operations, which allow to order them dynamically, thus providing a flexible scheduling of design operations.

Assuming that P is a process, that T1, T2 and Ti are design tasks, and that <cond> is a boolean expression involving tasks' input/output parameters, the operations include:

- start/suspend/resume/abort a process, e.g., START (P),
- start/suspend/resume/abort a task, e.g., START (T1),
- task sequencing, T1 ; T2,
- task parallelization, T1 / T2,
- task rendez-vous, T1 & T2,
- conditional loop, LOOP(T1, <cond>),
- conditional branching among partially ordered tasks, BRANCH(Ti, <cond>),
- unconditional backtracking among task execution sequences, BACK(Ti).

The last operation implements the imperative branching among tasks, i.e., the user-controlled flow of control among tasks. This supports the flexibility required by human decision-making during design work.

The previous two operations (conditional loop and branching) implement the trial and error processes.

The sequence and parallel operators implement the usual task control flow.

The rendez-vous implements task synchronization. This supports asynchronous tasks management, and delay handling.

The template provided by the process model can be modified on-line by the designers. Conditional branching and loop operations, as well as unconditional backtracking, can be inserted dynamically in an exising template, while executing predefined task sequences.
The stop and resume operators allow existing templates to be tested and modified, e.g., by inserting new conditional loops depending on intermediate parameter values.
The following example process P, defined by the partially ordered set of tasks T1, T2, ..., T6 :

$$P = (T1 ; (T2 / (T3 ; T4))\&(T5; LOOP(T6, cond1))$$

defines the template described in Figure 13.
Tasks need not be started on the same machine, allowing for distributed task management. Nor do they need to be launched simultaneously. This is why tasks T1 and T5 in Figure 13 are not linked together.
This conforms to concurrent engineering activities, which may be running independently for long periods of time, but have to meet at specified deadlines. For example, airframe design and engine design for aicraft are undertaken by totally different companies. But flight-testing and certification of the aircraft require the engines to be fitted on the airframe at a specified date, corresponding to the project schedule.

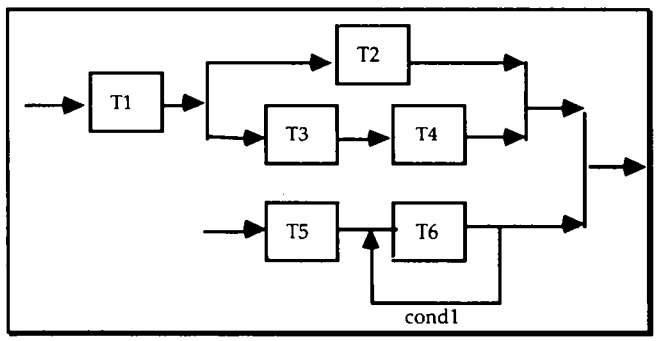

Figure 13. Process template.

6. Implementation

The process model is implemented using the object model in SHOOD, described in section 2. It is based on the extensive use of active rules for implementation of the operations and for scheduling the tasks [ORT95]. The tasks are modeled in turn by scripts which include sets of active rules. The latter are in charge of the activation of external basic operations in dedicated software, e.g., a geometric modeler.
The reflexive nature of the object model used allows for dynamic restructuring of the process models, because they are modeled by classes, which structure and attributes can be modified on-line [BOU95a].
The automaton described in section 4 is implemented in ARGOS [JOU94a, MAR90]. It is an imperative language for the specification and verification of

reactive systems. It is based on the hierarchical and parallel composition of automata. Unlike Statecharts, it has a sound and formally well defined semantics [GAR95, PNU91]. The full description of ARGOS is out of the scope of this paper. It is detailed extensively in [JOU94a, JOU94b].
Informally, automata are defined in ARGOS by states and transitions. The transitions are labelled by input and output events. Boolean event combinations trigger the transitions. Output events are produced when the transitions are fired. This allows the automata to communicate and to produce resulting events.
The ARGOS compiler produces several output formats, which can be input into several specification and verification software, e.g., ESTEREL [BER92, BOU91].
The automaton depicted in Figure 12, that implements the example described in Section 5, includes 46 states and 2425 transitions.
One advantage of ARGOS is that "observer" automata can be defined and implemented, using the same formalism as for the reactive object model.
These observers can detect illegal situations, for example incorrect requests by the designers, when the objects are in particular configurations (e.g., inconsistent objects cannot be made public). Incorrect interactions are also prevented in this way, when the objects are modified by simultaneous updates from the designers.

7. Conclusion

Concurrent engineering platforms emerge rapidly, as the competition for increased market shares drives the forces of manufacturing companies. Whereas much attention has been paid in recent years to software interoperability [GEN92a] and knowledge-level communication infrastructures [GEN92b], the need for increased capabilities in software integration platforms arises [NGU94]. This stems from the fact that more knowledge-level tools are required for the applications to cooperate efficiently.
Much has been written on concurrent engineering platforms, and feasibility demonstrators have been implemented. Yet, few practical tools have been proposed that impact on the effectiveness of concurrent engineering. This is even more crucial when considering complex composite objects that include multiple simultaneous representations [TEN92].
Merging existing technologies in open environments is probably a promising cross-fertilization approach. Our goal is to foster new developements that build on specification and verification techniques, together with research in knowledge sharing for advanced applications.
A reactive system approach is presented here, by which an automaton is designed as a plug-in addition to existing software integration platforms, that support concurrent engineering [GAR95]. The automaton implements a reactive model of the design objects.
It is intended to monitor on-line the impact of the decisions input by the designers, working concurrently on the parallel representations of the products. Together with the modeling of the global design process, the approach

followed here envisions the precise monitoring of change interactions among the design decisions. It uses the automaton as an on-line help, thus closely monitoring the designers' actions. It is intended to warn them of potentially hazardous decisions in real-time.
The automaton is implemented in ARGOS, an imperative language for reactive systems specification and verification.
Current work includes its integration to the platform SHOOD. SHOOD is designed to provide a knowledge-level integration infrastructure [NGU93]. It is based on a reflexive object model and a knowledge-level communication facility between the design tools. This includes a high-level communication protocol and the definition of application ontologies [BOU95], which are currently being addressed.

Acknowledgements

The project SHOOD is supported by INRIA, with contributions from Région Rhône-Alpes, and Pôle Productique Rhône-Alpes.
It is the result of long term research efforts from many people, among which are the following PhD students: Jérôme Boulenger who designed the interface with AutoCAD™, and is designing the ontology toolbox, called SHOT; Fethi Bounaas, who implemented the object model and the active rules; and Karine Duprez who is designing the intelligent help for a VLSI circuit simulator.
Chabane Djeraba, Assistant Professor at the University of Nantes, implemented the dependency relationships.
Last but not least, the support required for the implementation of the automaton in ARGOS is due to the invaluable help of Muriel Jourdan.

More information

The WWW server at INRIA provides extensive information on SHOOD, as well as the other INRIA projects, at: "http://www.inria.fr/".
A public archive of recent publications and information concerning the project SHOOD is available at: "ftp://ftp.imag.fr/pub/SHOOD/index.html".

References

[BER92] BERRY G., GONTHIER G. *The ESTEREL synchronous programming language: design, semantics, implementation.* In: Science of computer programming. 19(2). 1992.
[BOU95] BOULENGER J. *Semantic integration for knowledge sharing with SHOT : the SHOOD Ontology Toolbox*. NATO Workshop on Computer-Aided Logistics Support . Paris (France). November 1995.
[BOU95a] BOUNAAS F. *Using rules for object and schema evolution in an object-oriented system.* Proc. 17th Intl. Conf. TOOLS '95. Santa Barbara (USA). July 1995.
[BOU95b] BOUNAAS F., NGUYEN G.T. *Evolution in object-oriented systems: an approach using ECA rules.* Paper submitted for publication. 1995.
[BOU91] BOUSSINOT F., DE SIMONE R. *The ESTEREL language.* Proc. of the IEEE. (79)9. September 1991.
[BRO92] BROWN D.R, et al. *Next-Cut: a second generation framework for concurrent engineering*. In: Entreprise Modeling and Integration. C. Petrie (ed). McGraw-Hill. 1992.
[CUT93] CUTKOSKY M.R, et al. *PACT: an experiment in integrated concurrent engineering systems*. IEEE Computer. 26 (1). 1993.
[DER89] DERTOUZOS M., et al. *Made in America*. The MIT Press. Cambridge (Mass.). 1989.

[DJE93] DJERABA C. *Quelques liens sémantiques dans un système à base de connaissances.* Thèse de Doctorat. Université Claude Bernard. Lyon (France). December 1993.
[FIN92] FININ T., et al. *Specification of the KQML agent communication language.* The DARPA Knowledge Sharing Initiative. 1992.
[GAR95] GARG A., CHAWDRY P.K. *A case study in concurrent specification of reactive engineering systems.* Proc. "Concurrent Engineering: a Global Perspective" Conference. A.J. Paul, M. Sobolewski (eds). Concurrent Technologies Corp. Washington D.C. August 1995.
[GEN92a] GENESERETH M.R. *An agent-based framework for software interoperability.* Proc. DARPA Software Technology Conference. Arlington (USA). 1992.
[GEN92b] GENESERETH M.R., et al. *Knowedge interchange format. Version 3.0 reference manual.* Computer Science Dept. Stanford University. Tech. Report Logic-92-1. March 1992.
[GER93] GERO J.S. Proceedings of the IFIP WG 5.2 Workshop on "Formal Design Methods for Computer-Aided Design". Tallinn (Estonia). June 1993.
[GRU91] GRUBER T. *The role of common ontology in achieving sharable, reusable, knowledge bases.* In: Proc. 2nd Intl. Conf. Principles of knowledge representation and reasoning. Morgan-Kaufmann. 1991.
[GRU93] GRUBER T. *Toward principles for the design of ontologies used for knowledge sharing.* Proc. Intl. workshop on formal ontology in conceptual analysis and knowledge representation. Padova (Italy). 1993.
[JOU94a] JOURDAN M. *Etude d'un envi.onnement de programation et de vérification des systèmes réactifs, multi-langages et multi-outils.* PhD Thesis. Université Joseph Fourier. Grenoble (France). September 1994.
[JOU94b] JOURDAN M., MARANINCHI F. *A modular state/transition approach for programming reactive systems.* Proc. Workshop on Language, compiler, and tool support for real-time systems. Orlando (USA). June 1994.
[KIM90] KIM W. *Object-oriented databases: definitions and research directions.* IEEE Trans. on nowledge and Data Engineering. 2(3). 1990.
[MAR90] MARANINCHI F. *ARGOS: un langage graphique pour la conception, la description et la validation des systèmes réactifs.* PhD Thesis. Université Joseph Fourier. Grenoble (France). 1990.
[MAR93] MARANINCHI F., JOURDAN M. *Argos/Argonaute: une introduction.* Verimag-SPECTRE. 1993.
[MCG94] McGUIRE J., et al. *SHADE: technology for knowledge-based collaborative engineering.* In: Journal of Concurrent Engineering. 1(2). Sept. 1993.
[NEW82] NEWELL A. *The knowledge level.* Artificial Intelligence. 18(1). 1982.
[NGU91] NGUYEN G.T, et al. *Representing design objects.* In: Artificial Intelligence in Design '91. Butterworth-Heinemann. J.S Gero (ed.). 1991.
[NGU92a] NGUYEN G.T, et al. *Multiple object representations.* Proc. 20th ACM Computer Science Conference. May 1992. Kansas City (USA).
[NGU92b] NGUYEN G.T, et al. *SHOOD: A design object model* In: Artificial Intelligence in Design '92. Kluwer Academic Publ. J.S Gero (ed.). 1992.
[NGU93] NGUYEN G.T *SHOOD: plate-forme pour la conception assistée.* In: Ingénierie des systèmes d'information. Hermès (ed.) Vol. 1, n°3. 1993.
[NGU94] NGUYEN G.T, VERNADAT F. Cooperative information systems in integrated manufacturing environments. Proc. 2nd Intl. Conf. on Cooperative Information Systems. Toronto (Canada). May 1994.
[ORT95] ORTIZ R., DADAM P. *Towards the boundary of concurrency.* Proc. "Concurrent Engineering: a Global Perspective" Conference. A.J. Paul, M. Sobolewski (eds). Concurrent Technologies Corp. Washington D.C. August 1995.
[PAT92] PATIL F., et al. *The DARPA knowledge sharing effort: progress report.* Proc. 3rd Intl. Conf. on Principles of knowledge representation and reasoning. Morgan-Kaufmann. 1992.
[PET92] PETRIE C. *A minimalist model for coordination.* In: Entreprise Modeling and Integration. C. Petrie (ed). McGraw-Hill. 1992.
[PNU91] PNUELI A., SHALEV M. *What is in a step: on the semantics of Statecharts.* In: Lecture notes in computer science, 526. Theoretical aspects of computer science. 1991.
[TEN92] TENENBAUM J., et al. *Lessons from SHADE and PACT.* In: Entreprise Modeling and Integration. C. Petrie (ed). McGraw-Hill. 1992.

Unité de recherche INRIA Rhône-Alpes
46, avenue Félix Viallet - 38031 Grenoble Cedex 1 (France)

Unité de recherche INRIA Lorraine - Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes - IRISA, Campus universitaire de Beaulieu 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis - 2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

ISSN 0249 - 6399